Cassandra Crossing 657/ LLM, Algoritmi e Debito tecnologico

(657)—Mentre il termine “Vibe coding” perde per fortuna vigore, sempre più ambienti di programmazione “forzano” l’utilizzo di LLM per lo…

Cassandra Crossing 657/ LLM, Algoritmi e Debito tecnologico



(657)—Mentre il termine “Vibe coding” perde per fortuna vigore, sempre più ambienti di programmazione “forzano” l’utilizzo di LLM per lo sviluppo di software; cosa mai potrebbe andare storto che già altri non abbiano evidenziato?

26 gennaio 2026—Pare che la produzione di software tramite l’utilizzo di modelli linguistici sia in grande sviluppo; no, in tumultuosa crescita, anzi sia ormai divenuta inarrestabile ed indispensabile.

Definita inizialmente come “Vibe Coding”, è stata dapprima presentata come lasciapassare per chiunque volesse sviluppare software senza avere competenze di informatica e programmazione.

Poi, quando la cosa ha iniziato a sembrare l’idiozia che è, si sono invece osannati i vantaggi economici che l’impiego di questi metodi da parte di “veri” programmatori avrebbe consentito

alle aziende, aumentando la produttività dei programmatori esistenti; non è chiaro se dei senior che potevano fare a meno di una squadra di junior, oppure degli junior, che potevano scrivere software a livello di quello scritto dai senior.

Comunque certamente consentendo di tagliare posti di lavoro, presenti e futuri, facendo quindi scattare quell'automatismo che fa salire subito la quotazione in borsa di qualsiasi azienda.

Alla fine, i pareri di chi aveva provato davvero ad usare gli LLM in ambienti di produzione riguardo il risparmio di tempo e la qualità del codice prodotto hanno iniziato ad essere contrastanti, ed i primi dubbi ad essere presi sul serio.

Tuttavia lo sforzo di inserire a tutti i costi funzionalità guidate da LLM, comune a tutte le applicazioni commerciali, ha saturato di LLM anche tutti gli ambienti di sviluppo software.

E quindi tutti i programmatori, che lo volessero o no, si sono trovati ad **avere l'indice sul grilletto di una nuova "arma"**.

Alla vostra profetessa preferita non è dato sapere quanto l'utilizzo di questi ausili sia ufficialmente promosso, od addirittura richiesto, all'interno dei team di sviluppo aziendali. Nemmeno quanto sia il loro effettivo utilizzo, e se sia piuttosto iniziativa dei singoli programmatori pigri od in cerca di scorciatoie. E neanche quanto il suo uso nella scrittura di nuovo software sia percentualmente diffuso.

Ma, proprio dall'alto di questa piramide di ignoranza, la vostra profetessa preferita, il cui alter ego ha compiuto un lungo percorso proprio nello sviluppo "reale" di software nelle aziende, ritiene di poter esporre un parere informato, anzi di additare un vero pericolo che tuttavia pare non sia degno di discussione neppure tra gli addetti ai lavori … proprio come non lo fu un certo cavallo di legno.

Il problema è: **quanto "debito tecnologico" stiamo accumulando ed accumuleremo nelle applicazioni e nelle infrastrutture che mandano avanti la civiltà su questo pianeta**.

"*Debito tecnologico*" è il termine elegante che viene usato quando non si può dire "*la moltitudine di bug nascosti nel software di scarsissima qualità che viene normalmente prodotto dall'industria del software*".

Si sarebbero potuti utilizzare anche termini "scatologici", davvero inadatti a queste pagine. E poiché qui siamo tra signori, continueremo, anche se con fatica, ad usare solo il termine "*Debito tecnologico*".

Nel suo sempre più rapido affidarsi alle nuove tecnologie, ognuno di noi, che lo sappia o no, che ne sia cosciente o no, si affida continuamente a sistemi hardware/software che possono "tradirlo".

E non stiamo ancora parlando delle parole e delle istruzioni messe in fila a caso da un LLM, ma di "normali" programmi, sviluppati da esseri umani con metodologie più o meno adeguate e che, contenendo errori dovuti ad uno sviluppo software inadeguato, malfunzionano senza preavviso.

Dal cellulare che si pianta fino alla catastrofe planetaria per un collasso sistemico di reti informatiche e di distribuzione. Dall'azienda che fallisce per una procedura di backup errata fino al paziente oncologico a cui la macchina per radioterapia fa un buco nella testa o nel petto.

Tutte storie vere e già viste, accadute senza bisogno di ricorrere agli LLM per scrivere codice. Solo alla catastrofe planetaria ancora non siamo mai arrivati, ma eventi come l'"affaire" Maersk dovrebbero ricordarci quanto ci si può andar vicino, ed essere l'esempio pratico di quello che potrebbe succedere su scala molto più grande.

Ma terminiamo questa geremiade, che per i bene informati, in effetti, non contiene nessuna novità.

Il **debito tecnologico** rappresentato dalla massa di bug presenti nel software attuale esiste davvero, ed alcuni di questi bug causeranno, domani come ieri, il solito numero di danni e vittime.

Cosa potrebbe succedere se in pochi anni la maggior parte del codice nuovo o modificato fosse scritto da o tramite gli LLM? **Da dei programmi che non hanno nessun vincolo di realtà, ma solo di verosimiglianza**.

Da macchine apparentemente molto brave a scrivere sorgenti in maniera bella e credibile, grazie al fatto che i linguaggi di programmazione non sono complessi linguaggi naturali, ma linguaggi formali, dotati di grammatiche forti, chiuse e complete, ed anche al fatto che possono "copiare" a man bassa da Github e dagli altri repository di codice con cui sono state "addestrate".

Che errori possono essere generati da questo nuovo modo di "sviluppare" il software? E' banale chiedersi se saranno di più o di meno di quelli che sarebbero stati prodotti dalle modalità di sviluppo attuali.

Ma prevedere la quantità di errori è materia che interessa prevalentemente i consigli di amministrazione, che devono valutare l'ammontare dei danni che saranno chiamati a risarcire, ed il costo delle relative polizze di assicurazione. La maggiore o minore dimensione finanziaria del debito tecnologico che verrà così accumulato non è, secondo Cassandra, il vero problema.

La domanda che preoccupa Cassandra, e che a suo parere dovrebbe essere fatta per prima, è non "quanto" ma "come" sbaglieranno i programmi generati usando gli LLM. Che tipo di errori saranno contenuti nel codice da loro generato, o generato con il loro ausilio.

Saranno errori simili a quelli prodotti dagli umani, quasi sempre derivanti da un uso malaccorto delle primitive dei linguaggi di programmazione, o non piuttosto errori del tutto diversi, derivanti dalla non "comprensione" da parte degli LLM delle specifiche tecniche, del codice esistente e dei prompt?

Che il codice generato tramite LLM possa contenere più facilmente errori insidiosi, di "concetto" piuttosto che di programmazione malaccorta, sembrerebbe ragionevole, visto che il codice viene scritto in totale mancanza di comprensione, e senza nessun vincolo di realtà.

Questo potrebbe cambiare completamente la tipologia di debito tecnologico che accumuliamo, passando da errori che conosciamo e con cui abbiamo imparato a "convivere" ad altri che sono ignoti all'attuale "sistema immunitario" dell'industria del software.

Avremmo non software che contiene "normali" errori che portano a malfunzionamenti ed a deviazioni dal comportamento atteso, ma software i cui stessi algoritmi possono essere errati, ma in un modo sottile, e che non sarebbe evidenziabile con gli odierni strumenti di programmazione.

Un software i cui algoritmi non siano affidabili ma sottilmente difformi dalle specifiche potrebbe malfunzionare in maniera inusuale, non semplicemente fallendo nel suo compito, ma facendo cose diverse ed impreviste.

Per questo motivo, in fondo al luuuuungo elenco di problemi tecnici e sociali che l'utilizzo forzato di LLM senz'altro causerà o potrebbe causare, Cassandra suggerisce di aggiungere anche questo.

Scrivere a Cassandra—Twitter/X—Mastodon
Videorubrica "Quattro chiacchiere con Cassandra"

Lo Slog (Static Blog) di Cassandra
L'archivio di Cassandra: scuola, formazione e pensiero

Cassandra per i posteri: l'archivio su Internet Archive

Licenza d'utilizzo*: i contenuti di questo articolo, dove non diversamente indicato, sono sotto licenza Creative Commons Attribuzione—Condividi allo stesso modo 4.0 Internazionale (CC BY-SA 4.0),* tutte le informazioni di utilizzo del materiale sono disponibili a questo link.

By Marco A. L. Calamari on February 2, 2026.

Canonical link

Exported from Medium on March 30, 2026.